# CO223 - Lab 04
# Introduction to Socket Programming in Python

---

## Socket Programming

Socket programming is a way of connecting two nodes on a network to communicate with each other. One socket(node) listens on a particular port at an IP, while the other socket reaches out to the other to form a connection. The server forms the listener socket while the client reaches out to the server.
They are the real backbones behind web browsing. In simpler terms, there is a server and a client.

## Sockets

Sockets are the endpoints of a bidirectional communications channel. Sockets may communicate within a process, between processes on the same machine, or between processes on different continents.
Sockets may be implemented over a number of different channel types: Unix domain sockets, TCP, UDP, and so on. The socket library provides specific classes for handling the common transports as well as a generic interface for handling the rest.

## Terms used in Socket

- **Domain :**
    - Describes the  family of protocols that is used as the transport mechanism.
    - Contains constant values such as AF_INET, PF_INET, PF_UNIX, PF_X25 etc.

- **Type :**
    - The type of communications between the two endpoints
        - SOCK_STREAM for connection-oriented protocols
        - SOCK_DGRAM for connectionless protocols.

- **Protocol**
    - Used to identify a variant of a protocol within a domain and type
    - Typically zero.

- **Hostname**
    - The identifier of a network interface

- **Port**
    - Server listens for clients calling on one or more ports.
    - Can  be a Fixnum port number, a string containing a port number, or the name of a service.

## Server Socket Methods

- **s.bind() :** binds address (hostname, port number pair) to socket.
- **s.listen() :** sets up and starts a TCP listener.
- **s.accept() :** passively accept TCP client connection, waiting until connection arrives (blocking).

## Client Socket Methods

- **s.connect() :** initiates TCP server connection

## General Socket Methods

- **s.recv() :** receives TCP message
- **s.send() :** transmits TCP message
- **s.recvfrom() :** receives UDP message
- **s.sendto() :** transmits UDP message
- **s.close() :** closes socket
- **socket.gethostname() :** Returns the hostname.

## Creating a Socket

```
import socket

# create socket
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
```

The s variable is our TCP/IP socket. The AF_INET is in reference to the family or domain, it means ipv4, as opposed to ipv6 with AF_INET6.
The SOCK_STREAM means it will be a TCP socket, which is our type of socket.

A socket will be tied to some port on some host. In general, you will have either a client or a server type of entity or program.

In the case of the server, you will bind a socket to some port on the server (localhost). In the case of a client, you will connect a socket to that server, on the same port that the server-side code is using.

## Creating a Simple Server

For IP sockets, the address that we bind to is a tuple of the hostname and the port number.

```
Host = socket.gethostname()
Port = 1234
s.bind((Host, Port))
```

Next, let's listen for incoming connections. We can only handle one connection at a given time, so we want to allow for some sort of a queue, just in case we get a slight burst. If someone attempts to connect while the queue is full, they will be denied.

Let's make a queue of 5:

```
s.listen(5)
```

Then, the code to listen,

```
while True:
    # now our endpoint knows about the OTHER endpoint.
    clientsocket, address = s.accept()
    print(f"Connection from {address} has been established.")
```

Full code for server.py:

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
Host = socket.gethostname()
Port = 1234
s.bind((Host, Port))
s.listen(5)

while True:
    # now our endpoint knows about the OTHER endpoint.
    clientsocket, address = s.accept()
    print(f"Connection from {address} has been established.")
```

## Creating a Simple Client

Let us write a very simple client program which opens a connection to a given port 1234 and given host. This is very simple to create a socket client using Python's socket module function.

Now, since this is the client, rather than binding, we are going to connect.

```
s.connect((socket.gethostname(), 1234))
```

In the more traditional sense of client and server, you wouldn't actually have the client and server on the same machine. If you wanted to have two programs talking to each other locally, you could do this, but typically your client will more likely connect to some external server, using its public IP address, not socket.gethostname(). You will pass the string of the IP instead.

Full client.py code :

```
import socket

s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((socket.gethostname(), 1234))
```

Now, as you have full codes as server.py and client.py, you have to run the codes.
First run the server and then the client.

Now we have simply created a connection between the server and the client.

## Lab Exercise

1. Follow the instructions given above in the description and create the codes for sever.py and client.py. Include all the necessary comments for the code lines. Run both codes and observe the outputs.

2. The initial code is only for creating a connection between server and client. Modify the above codes to send a message from the server to the client. Client should display the message received.

3. An echo server is a simple server that receives a message from a client and sends the same message back to the client. Modify the above code to create an echo server.

## Submission

Create a report renamed as **E18XXX_Report.pdf** (XXX is your E Number).

The report should include,

- Answers for the lab exercise question 1 -3 with the necessary explanations, screenshots of the codes segments and the output
- Include comments in the code to explain

Submit a zipped folder named **E18XXX_Lab03.zip**including the report and the .py files for all the 3 tasks. (Include the codes for 3 tasks in separate sub folders named Task01, Task02, Task03)